**REST Person**

**api/person**

PATH complete

GET method

returns all persons in the database in json format

if the database is empty it throws a 204 error message

PATH complete/{id}

GET method

returns the person with the given id in json format

throws a 404 error if the person doesnt exist

throws a 400 in case of a wrong input (ex. non integer type)

PATH contactinfo

GET method

returns the contact info of all persons in json format

if the database is empty it throws a 204 error message

PATH contactinfo/{id}

GET method

returns the person with the given id in json format

throws a 404 error if the person doesnt exist

throws a 400 in case of a wrong input (ex. non integer type)

PATH phone/{phone}

GET method

returns the person with the given phone in json format

throws a 404 error if the person doesnt exist

throws a 400 in case of a wrong input (ex. non integer type)

PATH city/{city}

GET method

returns the persons that live in the given city in json format

throws a 404 error if there is no person in that city

PATH zip/{zipcode}

GET method

returns the persons that live in the given city in json format

throws a 404 error if there is no person in that city

PATH hobby/{hobby}

GET method

returns the persons that have the given hobby in json format

throws a 404 error if no person has that hobby

PATH hobby/count/{hobby}

GET method

returns the number of persons with the given hobby

PATH zip/count/{zipcode}

GET method

returns the number of persons that live in the city with the given zipcode

PATH street

GET method

returns all the streets in json format

PATH zip

GET method

returns all the zipcodes in json format

POST method

adds a new person

PUT method

edits a person

**REST Person**

**api/company**

PATH complete

GET method

returns all companies in the database in json format

if the database is empty it throws a 204 error message

PATH complete/phone/{number}

GET method

returns the company with the given phone in json format

throws a 404 error if the company doesnt exist

throws a 400 in case of a wrong input (ex. non integer type)

PATH complete/cvr/{cvr}

GET method

returns the company with the given cvr in json format

throws a 404 error if the company doesnt exist

throws a 400 in case of a wrong input (ex. non integer type)

PATH contactinfo

GET method

returns the contact info of all companies in json format

if the database is empty it throws a 204 error message

PATH id/{id}

GET method

returns the company with the given id in json format

throws a 404 error if the company doesnt exist

throws a 400 in case of a wrong input (ex. non integer type)

PATH cvr/{cvr}

GET method

returns the company with the given cvr in json format

throws a 404 error if the company doesnt exist

throws a 400 in case of a wrong input (ex. non integer type)

PATH city/{zipcode}

GET method

returns the companies in the city with the given zipcode in json format

throws a 404 error if the company doesnt exist

throws a 400 in case of a wrong input (ex. non integer type)

PATH count/{zipcode}

GET method

returns the number of companies in the city with the given zipcode

throws a 404 error if the company doesnt exist

throws a 400 in case of a wrong input (ex. non integer type)

PATH street

GET method

returns all the streets in json format

throws a 204 error if there are no streets

PAT zip

GET method

returns all the zipcodes

throws a 204 error if there are no zip codes

PATH employess/more/{min}

GET method

returns the companies that have less employess than the given number in json format

throws a 404 error if the company doesnt exist

PUT method

edit a company

throws a 404 error if you try to edit a non existing company

POST method

create a company

throws a 400 error if you dont fill all the fields

PATH {id}

DELETE method

delete the company with the given id

throws a 404 error if the company doesnt exist

throws a 400 in case of a wrong input (ex. non integer type)

**Inheritance Strategy**

We used the Single table strategy because is simpler.

The single table strategy maps all entities of the inheritance structure to the same database table.

This approach makes polymorphic queries very efficient and provides best performance,

but it also has some drawbacks.The attributes of all entities are mapped to the same database

table.Each record uses only a subset of the available columns and sets the rest of them to null.

SO you can not use not null constraints on any column that isnt mapped to all entities.

That can create data integrity issues.

**Testing Strategy**

The test are in the folder "Test Packages".The "IntegrationTest" package contains the integration tests

and the "UnitTest" package the unit tests for Company facade and the Person facade.

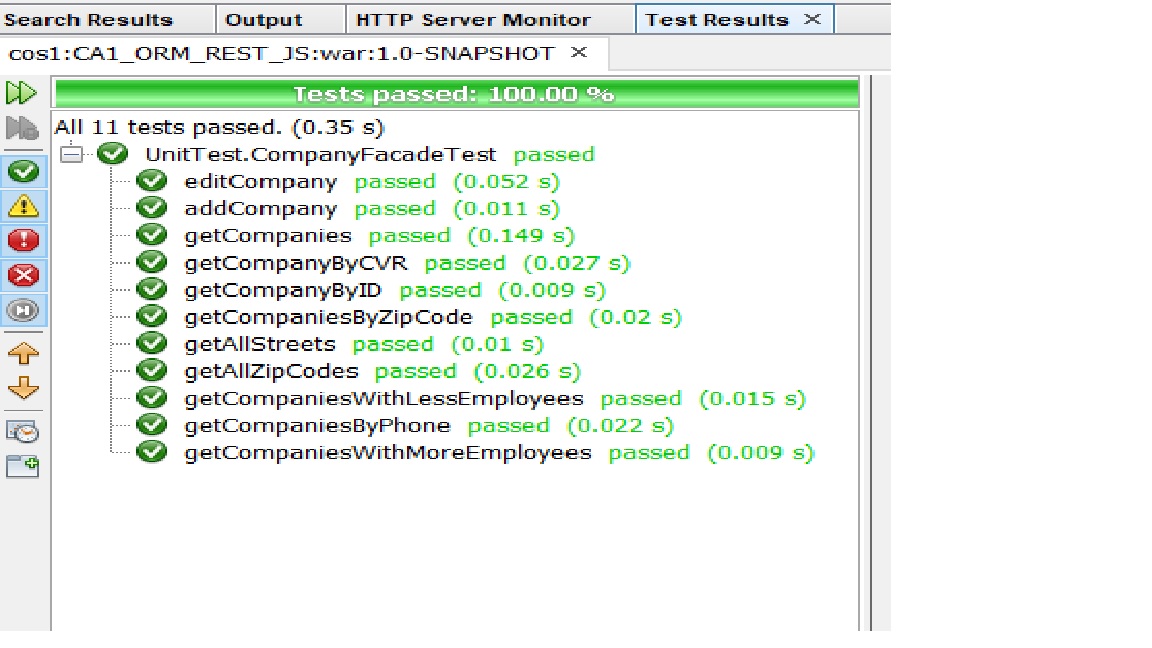
First we did manual testing with the end points

"UnitTest" package the unit tests for Company facade and the Person facade.

"IntegrationTest" package is used for testing the database facades.

We manually test the endpoints of our REST API.

**Unit Test Results for company facade**



**Unit Test results for Person Facade**

